FIG. 1

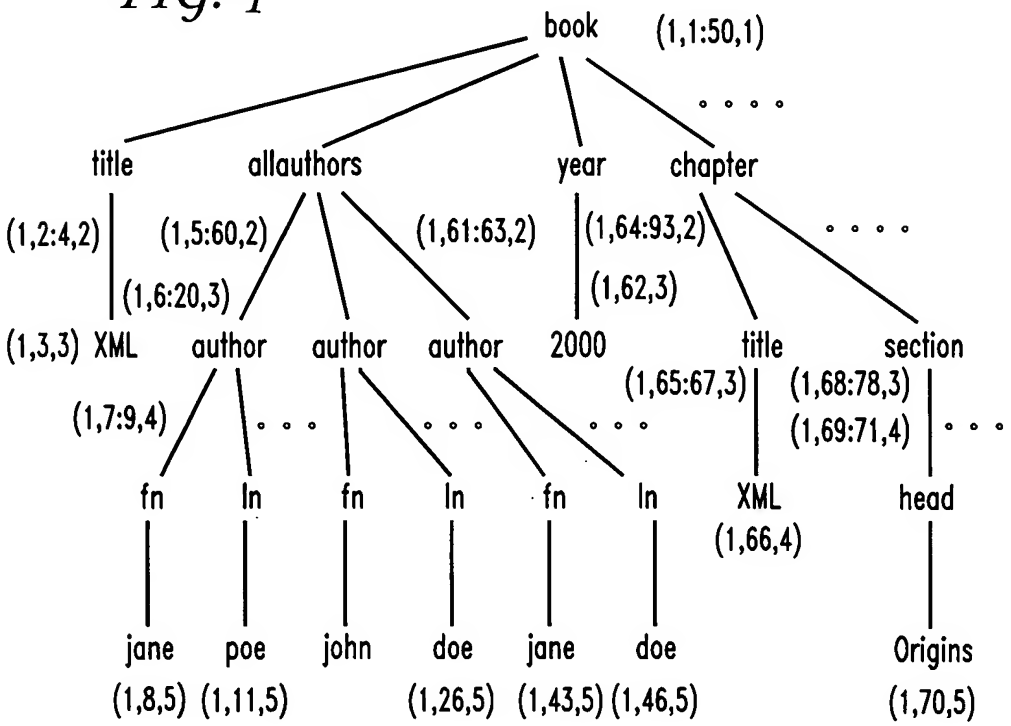


FIG. 2A

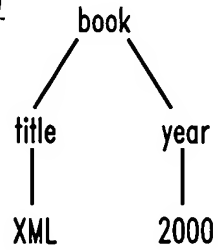


FIG. 2B

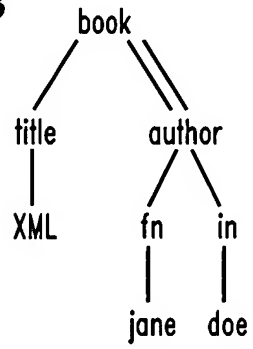


FIG. 3A

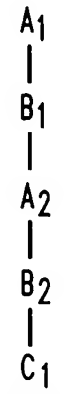


FIG. 3B

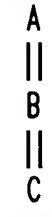


FIG. 3C

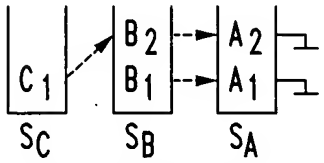


FIG. 3D

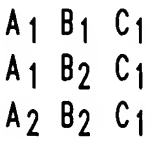


FIG. 4

```
Algorithm PathStack(q)
01  while ¬end(q)
02    qmin = getMinSource(q)
03    for qi in subtreeNodes(q)  // clean stacks
04      while (¬empty(Sqi) ∧ topB(Sqi) < nextL(Tqmin))
05        pop(Sqi)
06    moveStreamToStack(Tqmin, Sqmin, pointer to
                                top(Sparent(qmin)))
07    if (isleaf(qmin))
08      showSolutions(Sqmin,1)
09      pop (Sqmin)
Function end(q)
   return ∀qi ∈ subtreeNodes(q): isLeaf(qi) ⇒ eof(Tqi)

Function getMinSource(q)
   return qi ∈ subtreeNodes(q) such that nextL(Tqi)
     is minimal
Procedure moveStreamToStack(Tq,Sq,p)
01  push(Sq,(next(Tq),p))
02  advance(Tq)
```

PathStack

FIG. 4A

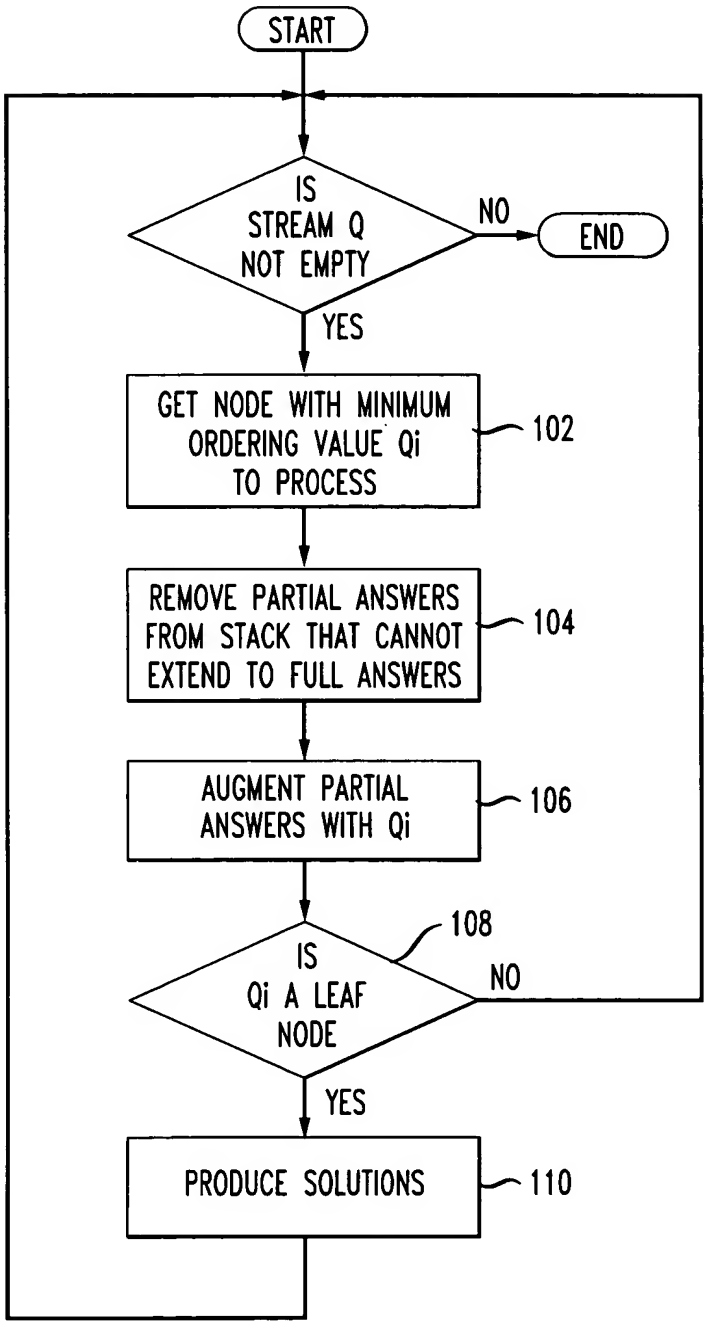

START
IS STREAM Q NOT EMPTY
NO
END
YES
GET NODE WITH MINIMUM ORDERING VALUE Qi TO PROCESS
102
REMOVE PARTIAL ANSWERS FROM STACK THAT CANNOT EXTEND TO FULL ANSWERS
104
AUGMENT PARTIAL ANSWERS WITH Qi
106
108
IS Qi A LEAF NODE
NO
YES
PRODUCE SOLUTIONS
110

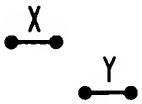
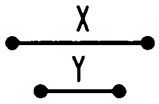
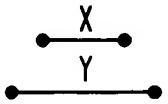
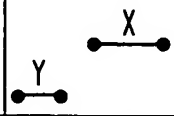
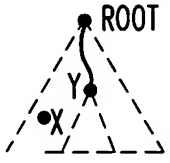
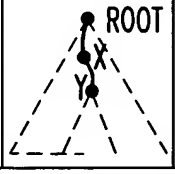
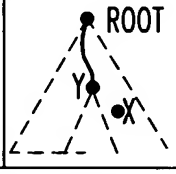
FIG. 5

```
Procedure showSolutions(SN,SP)
// Assume, for simplicity, that the stack of the query
// nodes from the root to the current leaf node we
// are interested in can be accessed as S[1],...,S[n].
// Also assume that we have a global array index[1..n]
//    of pointers to the stack elements.
// index[i] represents the position in the i'th stack that
// we are interested in for the current solution, where
// the bottom of each stack has position 1

// Mark we are interested in position SP of stack SN.
01 index[SN] = SP
02 if (SN == 1) // we are in the root
03    // output solutions from the stacks
04    output (S[n].index[n],...,S[1].index[1])
05  else // recursive call
06     for i = 1 to S[SN].index[SN].pointer_to_parent
07        showSolutions(SN - 1,i)
```

Procedure showSolutions

FIG. 6

	CASE 1	CASE 1	CASE 1	CASE 1
PROPERTY	X.R<Y.L	X.L<Y.L X.R>Y.R	X.L>Y.L X.R<Y.R	X.L>Y.R
SEGMENTS				
TREE				

CASES FOR PathStack AND TwigStack

FIG. 7

```
Algorithm PathMPMJ(q)
01 while (¬eof(Tq)∧ (isRoot(q)∨
                        nextL(q) <nextR(parent(q))))
02   for (qi ∈ subtreeNodes(q)) // advance descendants
03     while (nextL(qi) < nextL(parent)qi)))
04       advance(Tqi)
05     PushMark(Tqi)
06   if (isLeaf(q)) // solution in the streams' heads
       outputSolution()
07   else PathMPMJ(child(q))
08   advance(Tq)
09   for (qi ∈ subtreeNodes(q)) // backtrack descendants
10     PopMark(Tqi)
```

PathMPMJ

FIG. 8

```
Algorithm TwigStack(q)
   // Phase 1
01  while ¬end(q)
02    q_act = getNext(q)
03    if (¬isRoot(q_act))
04      cleanStack(parent(act), next(q_act))
05    if (isRoot(q_act) V ¬empty(Sparent(q_act)))
06      cleanStack(q_act, next(q_act))
07      moveStreamToStack(Tq_act, pointer to
                                top(Sparent(q_act)))
08      if (isLeaf(q_act))
09        showSolutionWithBlocking(Sq_act, 1)
10        pop(Sq_act)
11    else advance(Tq_act)
     // Phase 2
12  mergeAllPathSolutions()

Function getNext(q)
01  if (isLeaf(q) return q
02  for q_i in children(q)
03    n_i = getNext(q_i)
04    if (n_i ≠ q_i) return n_i
05  nmin = minarg_ni nextL(T_ni)
06  nmax = maxrarg_ni nextL(T_ni)
07  while (nextR(Tq) < nextL(T_nmax))
08     advance(Tq)
09  if (nextL(Tq) < nextL(T_nmin)) return q
10  else return nmin

Procedure cleanStack(S, actL)
01  while (¬empty(S) ∧ (topR(S) < actL))
02     pop(S)
```

TwigStack

FIG. 8A

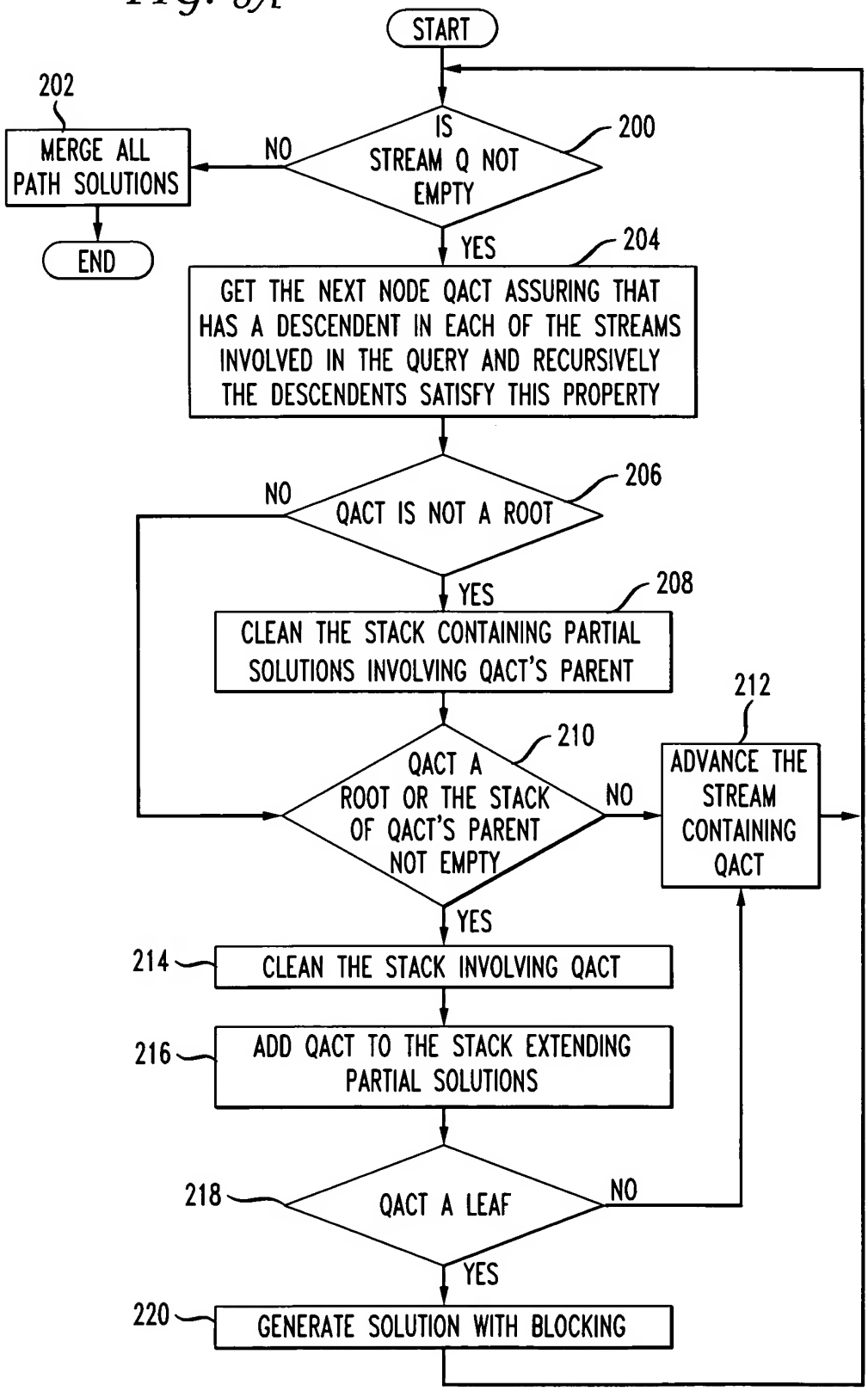

START
IS STREAM Q NOT EMPTY
200
NO
MERGE ALL PATH SOLUTIONS
202
END
YES
204
GET THE NEXT NODE QACT ASSURING THAT HAS A DESCENDENT IN EACH OF THE STREAMS INVOLVED IN THE QUERY AND RECURSIVELY THE DESCENDENTS SATISFY THIS PROPERTY
NO
QACT IS NOT A ROOT
206
YES
208
CLEAN THE STACK CONTAINING PARTIAL SOLUTIONS INVOLVING QACT'S PARENT
212
ADVANCE THE STREAM CONTAINING QACT
210
QACT A ROOT OR THE STACK OF QACT'S PARENT NOT EMPTY
NO
YES
214
CLEAN THE STACK INVOLVING QACT
216
ADD QACT TO THE STACK EXTENDING PARTIAL SOLUTIONS
218
QACT A LEAF
NO
YES
220
GENERATE SOLUTION WITH BLOCKING

FIG. 9

```
Algorithm TwigStackXB(q)
01  while ¬end(q)
02    q_act = getNext(q)
(03)  if (isPlainValue(Tq_act))
04      if (¬ isRoot(q_act))
05        cleanStack(parent(q_act), next(q_act))
06      if (isRoot(q_act) V ¬empty(S_parent(q_act)))
07        cleanStack(q_act, next(q_act))
08        moveStreamToStack(Tq_act, pointer to
                                top(S_parent(q_act)))
09        if (isLeaf(q_act))
10          showSolutionsWithBlocking(Sq_act,1)
11          pop(Sq_act)
12      else advance(Tq_act)
(13)  else  if (¬ isRoot(q_act) ∧ empty(Sparent(q_act)) ∧
              nextL(Tparent(q_act)) > nextR(Tq_act))
(14)  advance(Tq_act) // Not part of a solution
(15)  else // Might have a child in some solution
(16)  drillDown(Tq_act)
    // Phase 2
17  mergeAllPathSolutions()

Function getNext(q)
01  if (isLeaf(q) return q
02  for q_i in children(q)
03    n_i = getNext(q_i)
(04)  if (q_i ≠ n_i V ¬ isPlainValue(Tn_i)) return n_i
05  n_min = minarg_ni nextL(Tn_i)
06  n_max = maxrarg_ni nextL(Tn_i)
07  while (nextR(Tq) < nextL(Tn_max))
08    advance(Tq)
09  if (nextL(Tq) < nextL(Tn_min)) return q
10  else return n_min

Procedure cleanStack(S, actL)
01  while (¬ empty(S) ∧ (topR(S) < actL))
02    pop(S)
```

TwigStack

FIG. 9A

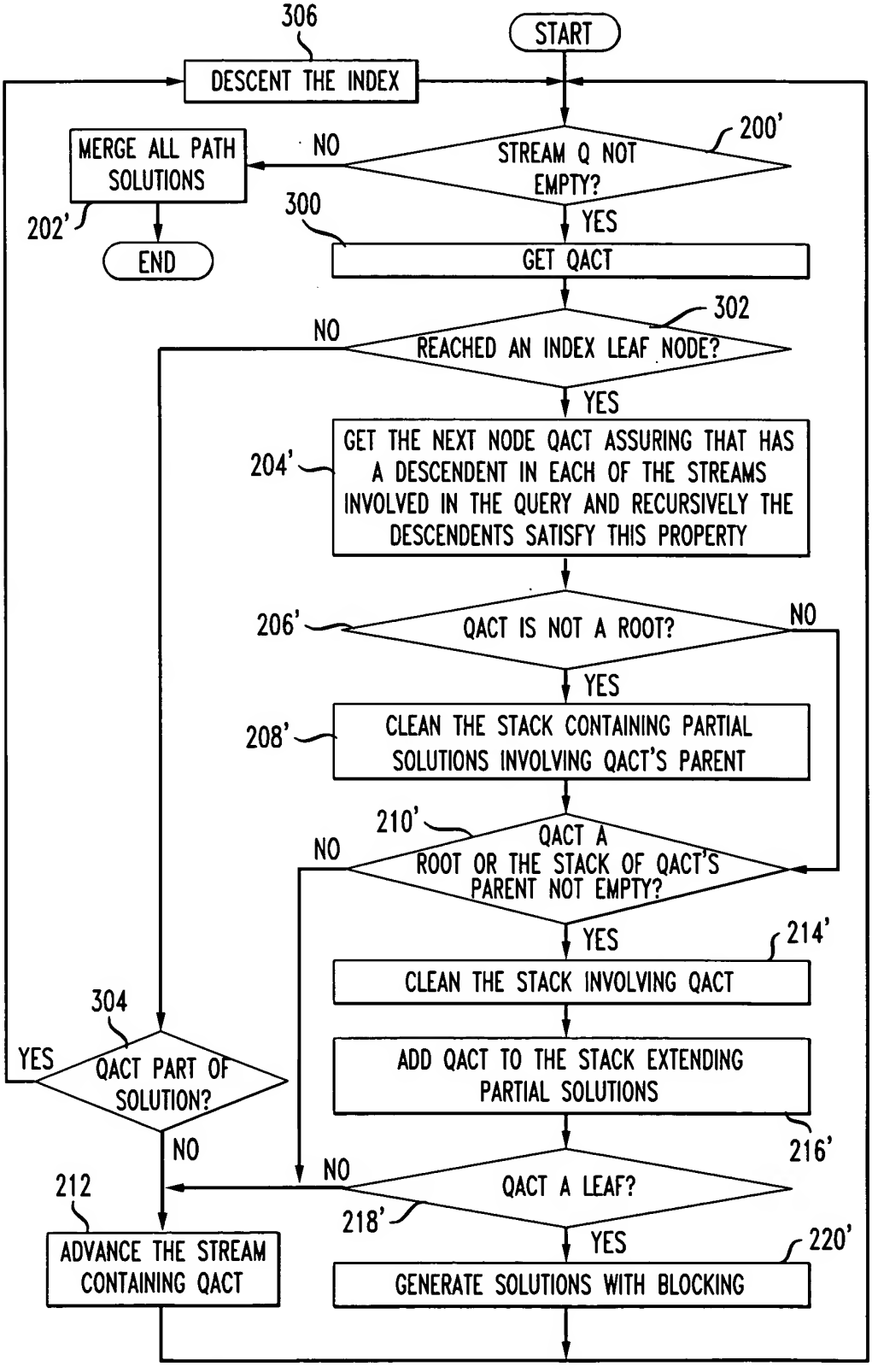

306
START
DESCENT THE INDEX
200'
MERGE ALL PATH SOLUTIONS
NO
STREAM Q NOT EMPTY?
202'
300
YES
END
GET QACT
302
NO
REACHED AN INDEX LEAF NODE?
YES
204'
GET THE NEXT NODE QACT ASSURING THAT HAS A DESCENDENT IN EACH OF THE STREAMS INVOLVED IN THE QUERY AND RECURSIVELY THE DESCENDENTS SATISFY THIS PROPERTY
206'
QACT IS NOT A ROOT?
NO
YES
208'
CLEAN THE STACK CONTAINING PARTIAL SOLUTIONS INVOLVING QACT'S PARENT
210'
NO
QACT A ROOT OR THE STACK OF QACT'S PARENT NOT EMPTY?
YES
214'
CLEAN THE STACK INVOLVING QACT
304
ADD QACT TO THE STACK EXTENDING PARTIAL SOLUTIONS
YES
QACT PART OF SOLUTION?
216'
NO
NO
212
QACT A LEAF?
218'
YES
220'
ADVANCE THE STREAM CONTAINING QACT
GENERATE SOLUTIONS WITH BLOCKING

FIG. 10

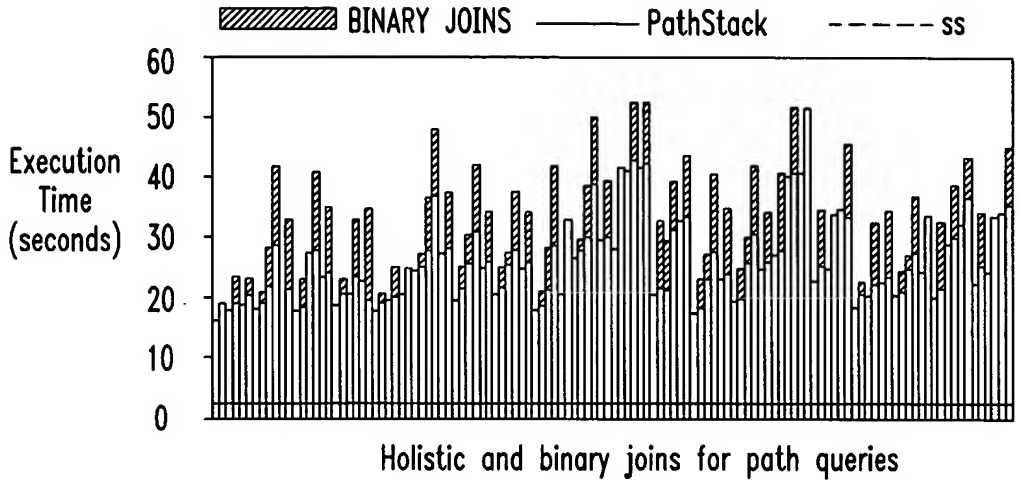

BINARY JOINS
PathStack
ss
Execution Time (seconds)
60
50
40
30
20
10
0
Holistic and binary joins for path queries

FIG. 11

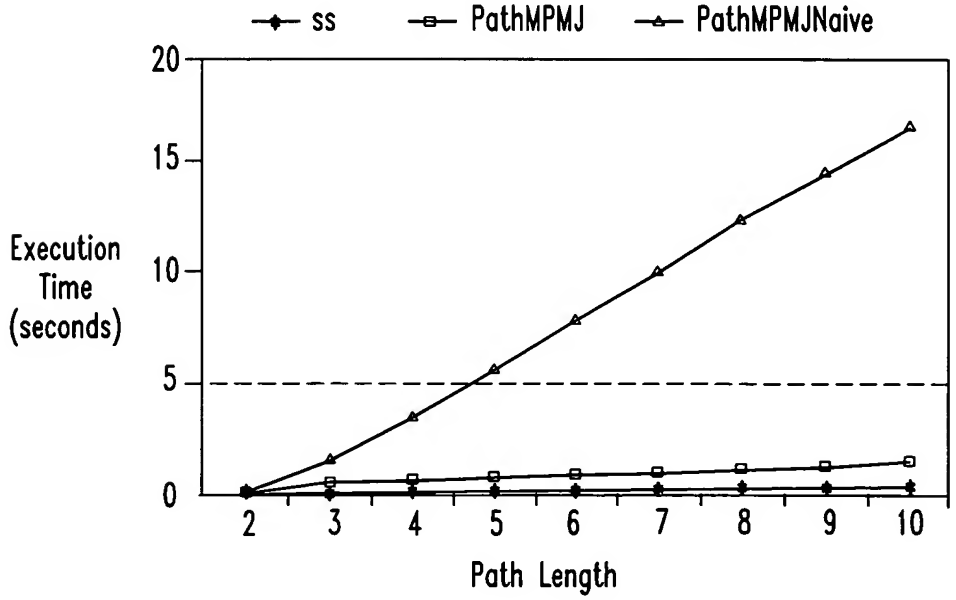

ss
PathMPMJ
PathMPMJNaive
Execution Time (seconds)
20
15
10
5
0
2
3
4
5
6
7
8
9
10
Path Length

FIG. 12A

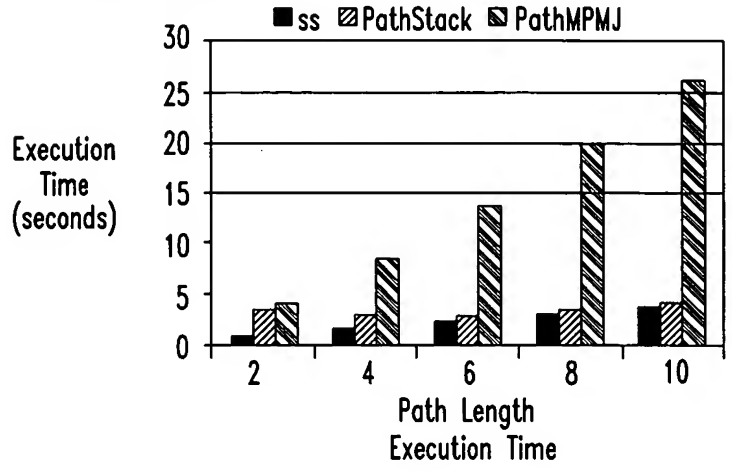


FIG. 12B

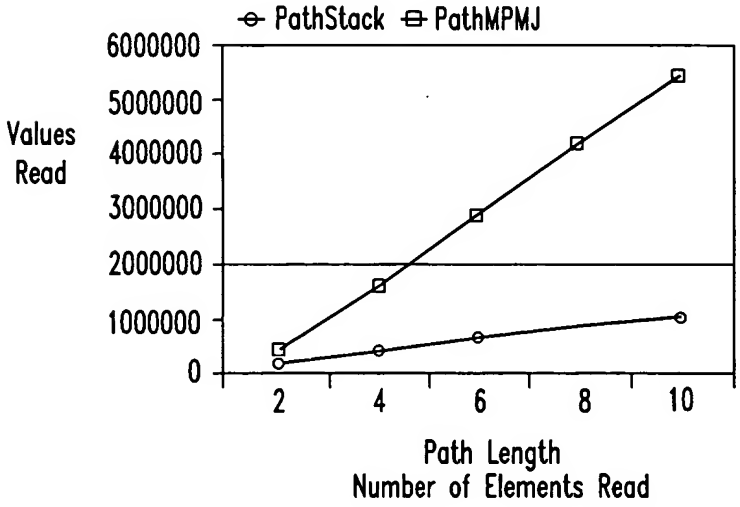


FIG. 13A

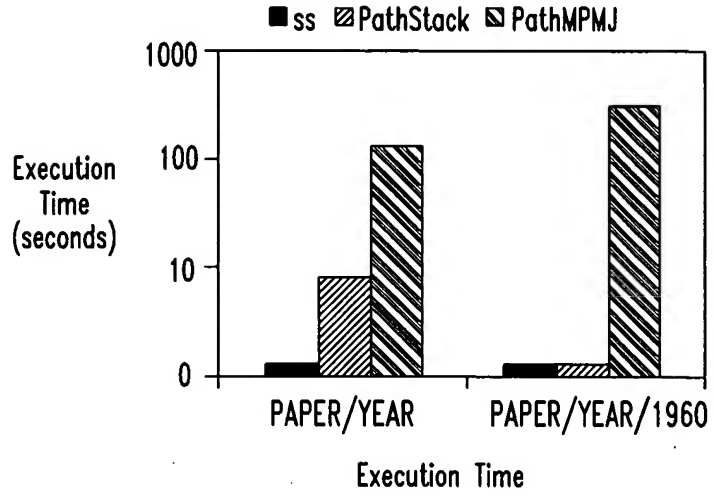


FIG. 13B

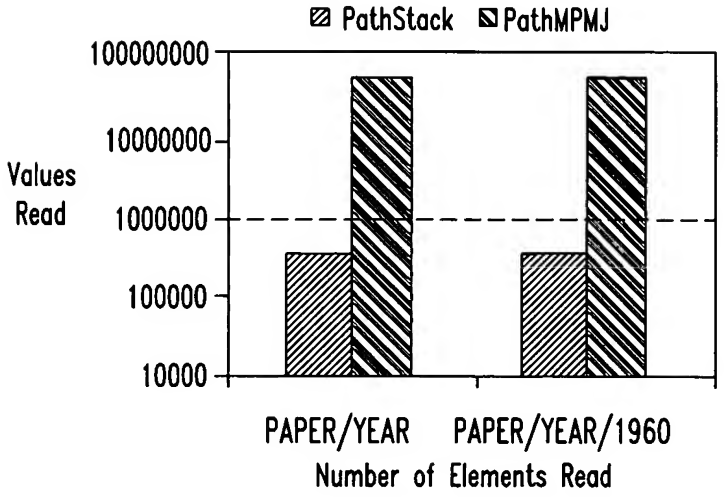


FIG. 14A

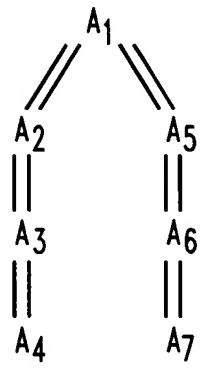


FIG. 14B

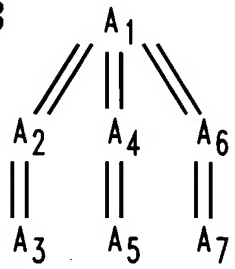


FIG. 14C

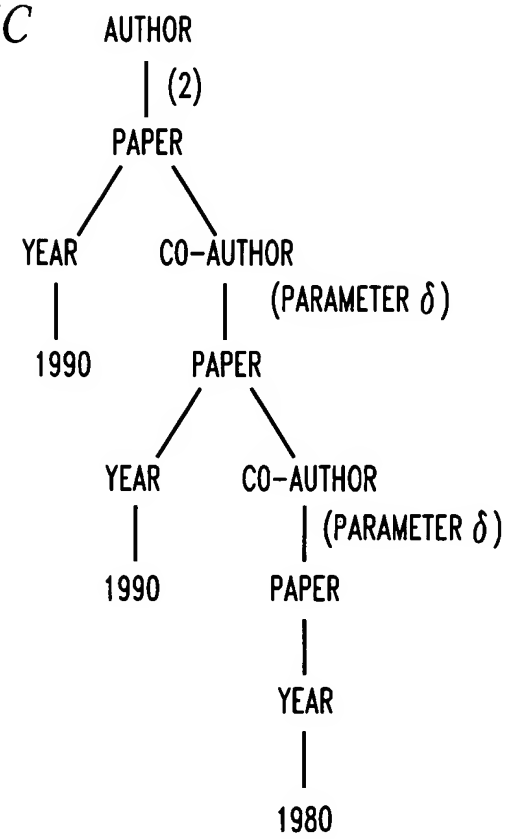


FIG. 15A

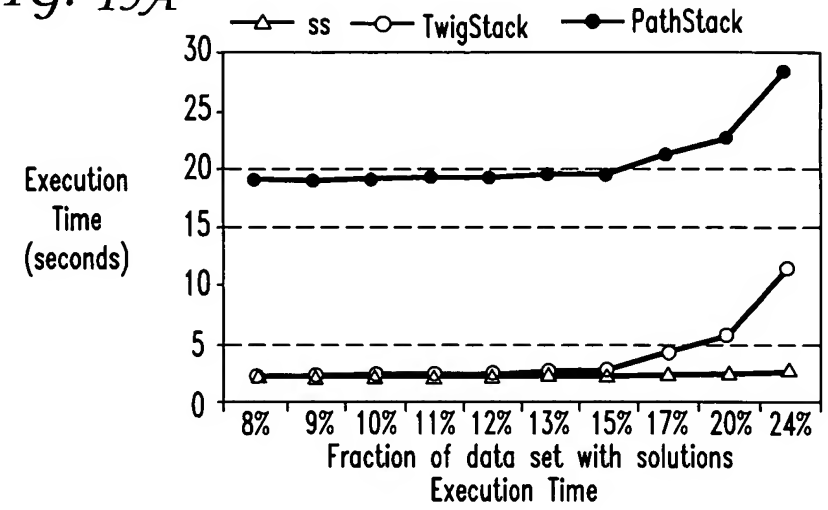


FIG. 15B

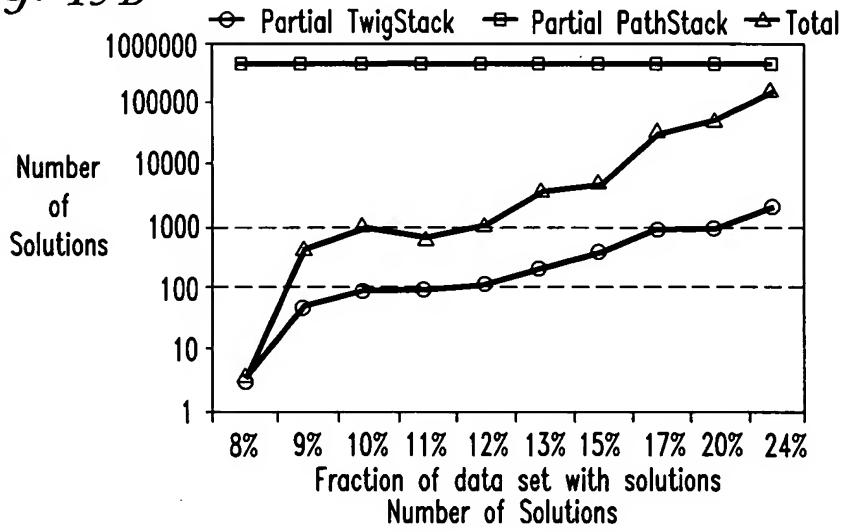


FIG. 15C

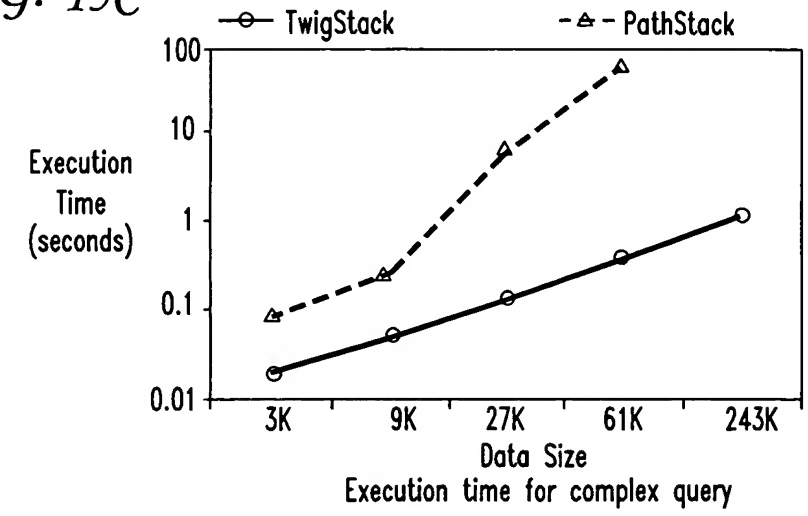


FIG. 16A

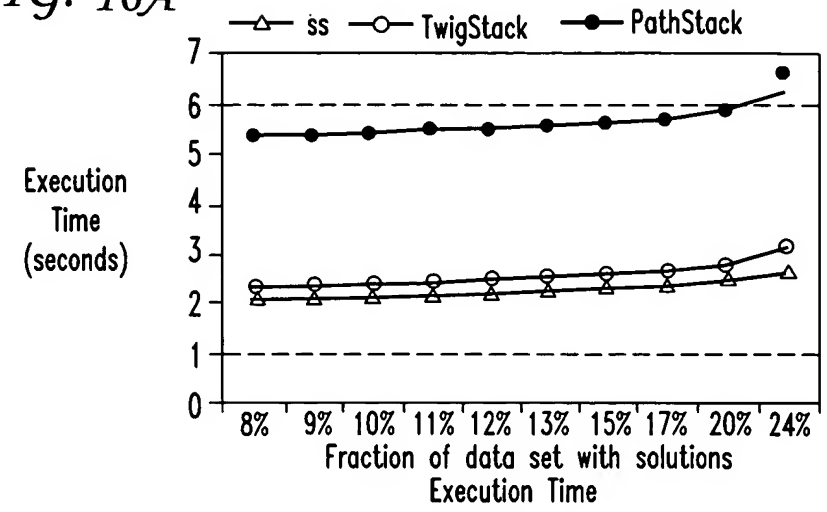


FIG. 16B

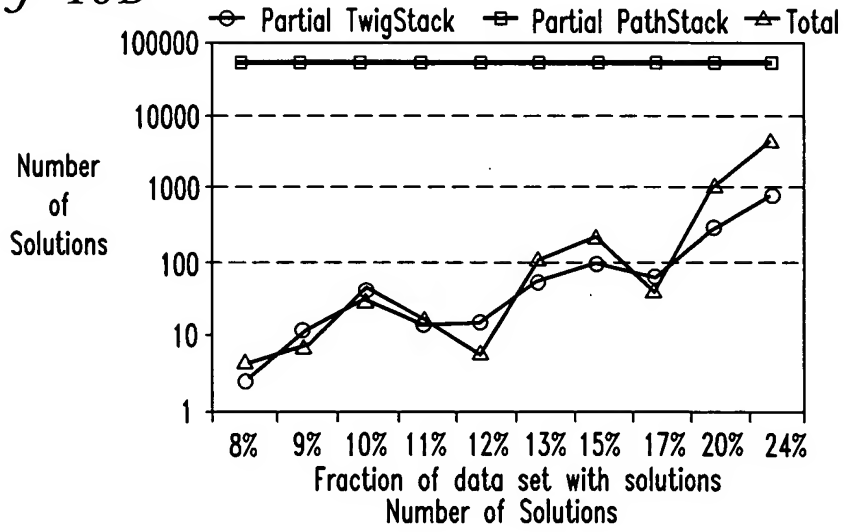


FIG. 16C

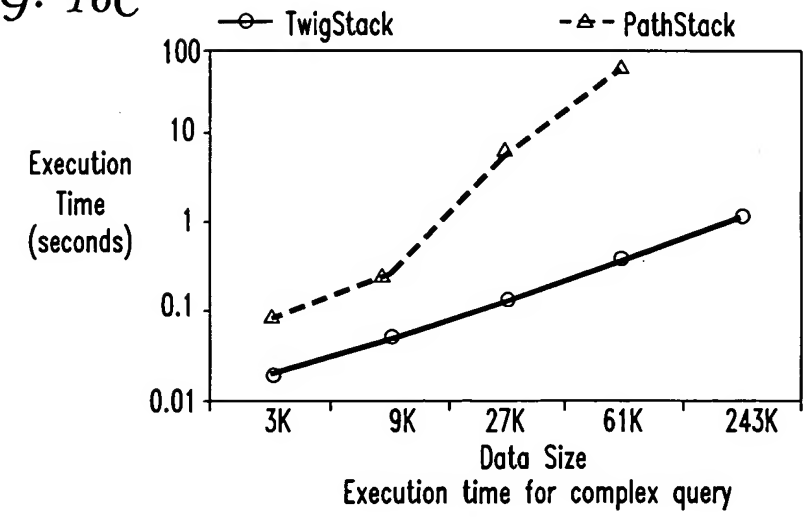


FIG. 17A

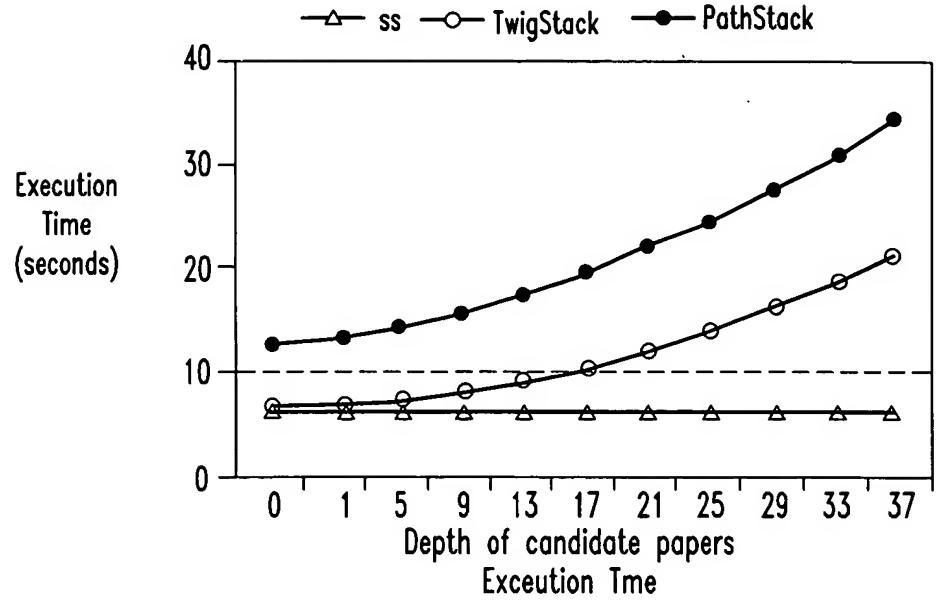


FIG. 17B

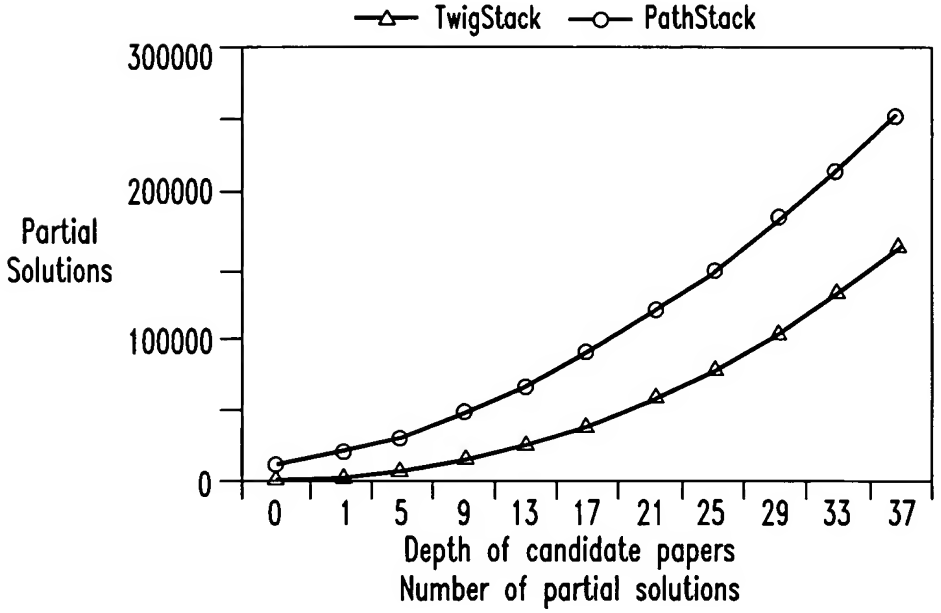


FIG. 18A

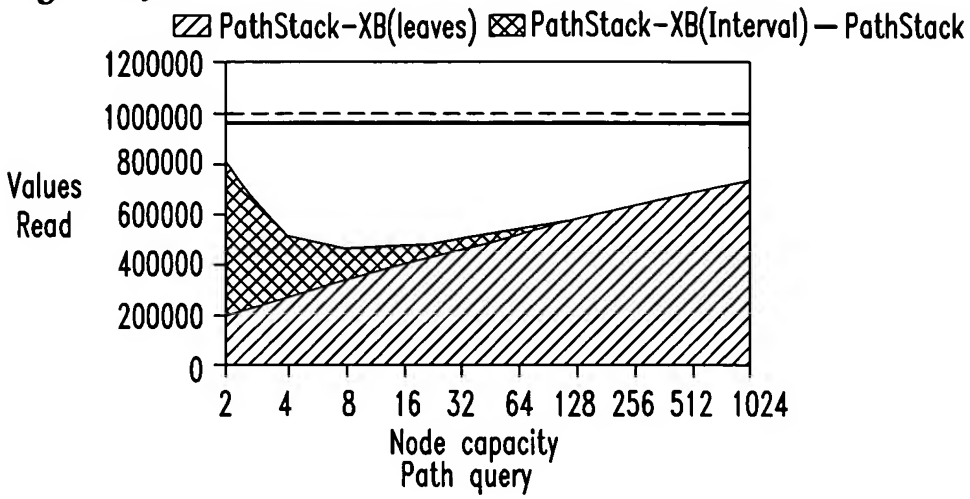


FIG. 18B

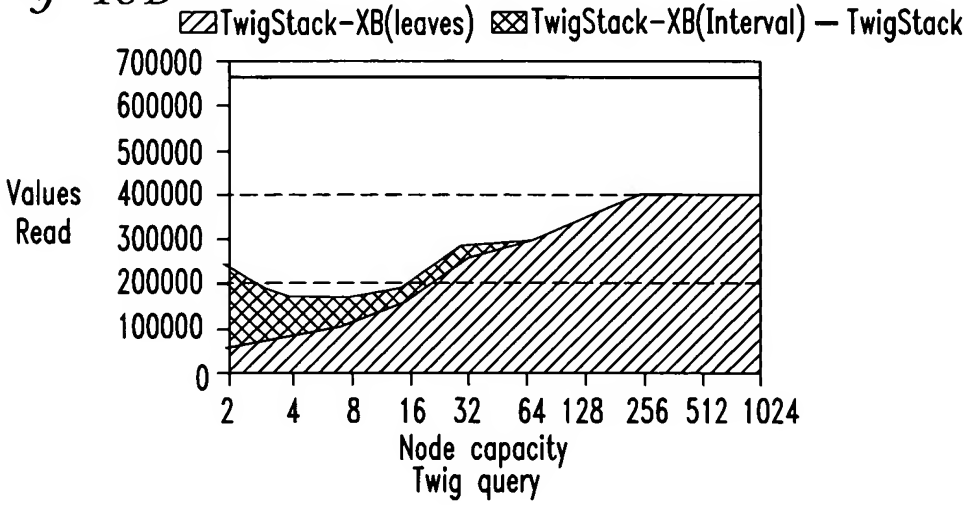


FIG. 18C